**SET – 1 Answers**

**1 Answer:**

The basic structure of a C program looks as follows:

Documentation section

Link section

Definition section

Global declaration section

main() function section{

Declaration part

Executable part

}

Subprogram section

Function1

Function2

:

Function n

**Explanation of section in the C program:**

* The Documentation section contains the meta-data (program description – problem statement, logic and implementation, author, date created, etc.) of the written program that follows. The description in the documentation section is placed within de-limiters (**/\***description **\*/** ). In addition, we can include comments, within the delimiters, throughout the program which can be used to explain the lines or logic behind them.
* Link section is used to add header files that are needed for the compiler to execute programs. These header files are from the predefined and written programs from the system library.
* Definition Section involves defining and initialization of the symbolic constants used in the program
* Global Declaration section is where global variables - used in main and one or more functions, are defined.
* **main()** function is where the C program starts compilation process, therefore every C program has a main function and it generally has two parts. Declaration part and Executable part. In declaration part, the variables that are used in the executable part are declared. The program execution starts at the opening ‘{‘and logically ends at the ‘}’. Each statement in the program must end with a semi-colon ‘;’
* **subprogram section** includes user-defined functions that are called in the main function to do some sub-tasks. usually these are written after main program by convention.

Except the main() function the other parts of a C program may or may not be included as per the requirements.

**C program to find the area of a circle**:

#include <stdio.h> // standard input output library file access //

**/\***The program calculates and prints the area of a circle whose radius is taken from the user input

author name

date of creation**\*/ //** Documentation

# define PI 3.14 // definition section

int main() { // main program starting

float radius, area; // variable declaration

printf(“Enter the radius of the circle :”);

scanf(“%f”, &radius);

area = PI \* radius \* radius;

printf(“The area of the circle is : %f”, &area);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2 Answer:**

The control statements in C program are programming concept on how to control the flow of execution of the program as usually the program statements are executed in sequence. The control statements determine the order in which statements are executed based on certain conditions. C provides several control statements that can be used to make decisions, perform loops, and alter the normal flow of execution.



The control statements in C in are:

1. **goto statement:** The goto statement looks like the following:



**goto** label;

**--------------** /\* intermediate statements \*/



**--------------**

label: /\* label is a valid variable name followed by colon\*/



statement;

the goto statement allows one to transfer control to a labelled statement within the same function. It provides a way to jump to a specific point in the program, bypassing any statements in between.The goto statement can jump forward or backward depending on the placement of the label.

1. The **if -statements:** The general structure of the if statements has a conditional expression which could be either true of false. The block of program inside the if block is executed only when the if condition is passed.

There are majorly three variants of if statements:

* **if -statement**: The if statement is used to execute a block of code if a certain condition is true. If the condition is false, the if block is skipped. it looks as follows:

if (condition){ /\* the ‘if’ block is entered only when the condition is true statement \*/

statement;

------------;

}

* **if-else statement:** The if-else statement allows to execute different blocks of code based on whether a condition is true or false. If the condition in the if statement is true, the corresponding code block is executed. Otherwise, the code block in the else statement is executed. The if-else block looks like :

if (expression) {

execute this part of block if expression is true

}else {

execute this part of block if expression is false

}

* Nested **if-else statement:** The if-else blocks can be nested, meaning we can have as many if-else blocks inside other if-else blocks. This can be used to perform multiple hierarchical or complex decision making. The code block looks like:

if (expression1) {

// code to be executed if expression1 is true

} else if (expression2) {

// code to be executed if expression2 is true

} else {

// code to be executed if both expression1 and expression1 are false

}

1. **The switch statement:** Switch case statements are a very good alternative to multiple if-else blocks as the nesting reduces simple passing of cases to execute a block of statements. The switch case looks like the following:

switch (expression) {

case constant1:

// code to be executed if expression matches constant1

break;

case constant2:

// code to be executed if expression matches constant2

break;

// more cases...

default:

// code to be executed if expression doesn't match any case

}

The expression is typically a variable with integral value. The code block that matches the case of the expression gets executed. If we don’t want to check the further cases, we can end the case blocks with break; statements to come out the switch block of program. A good practice is to use an optional default case, which gets executed when no cases get executed. This helps us to avoid the unexpected behaviour of the program.

1. **The conditional expression (? :) –** that looks like ---

Expression1? Expression2: Expression3.Here if the expression1 is true then the expression2 get executed else expression3 gets executed. This a short way to express simple if-else block statements.

1. **The while – loop:** The while loop is used to repeatedly execute a block of code as long as a condition is true. Before each iteration, the condition is checked, and if it evaluates to true, the code block is executed. The code looks like:

**while(**condition is true**){**

while loop body // the loop is exited when the condition is false

**}**

1. **Do-while loop**: The do-while loop is similar to the while loop, but the condition is checked at the end of the loop iteration. This guarantees that the code block is executed at least once. The code looks like:

do {

loop body

} while (condition);

1. **The for-loop**: The for loop is used to execute a block of code repeatedly for a specified number of times. It consists of three parts: initialization, condition, and update. The initialization is executed once at the beginning, the condition is checked before each iteration, and the update is executed at the end of each iteration. The for loop looks like:

**for(**exp1; exp2; exp3**){ /\*** exp1 is initialisation, exp2 is condition, exp3 is to update\*/

for-loop body /\*executes as long as the condition is true \*/

**}**

Nested for loops can be constructed within loops based on the requirements of the problem. These are loops executed inside loops.

1. **The break and continue statements:** The break statement is used to terminate the execution of a loop or switch statement immediately. When encountered, it exits the innermost loop or switch statement and continues with the code after the loop or switch. In contrast, the continue statement is used to skip the rest of the code inside a loop for the current iteration. When encountered, it goes to the next iteration of the loop without executing the remaining statements.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3 Answer:**

**Storage classes in C** are used to define the lifetime, scope, and visibility of variables or functions in a C program. They determine how memory is allocated and deallocated for variables and control the accessibility of variables and functions within different parts of a program. The main purpose of storage classes in C is to provide flexibility and control over the storage and usage of variables and functions. The different storage classes are automatic or local variables, global variables, static variables and external variables and their purposes are as follows:

**Automatic (or local) variables:** Automatic variables are declared within a function or block and have local scope, meaning they are accessible only within that specific function or block. They are automatically allocated memory when the function or block is entered and deallocated when it is exited. Automatic variables have a short lifetime, and each function call creates a new instance of these variables. They are the default storage class and are used for temporary storage and intermediate calculations within functions.

**Global variables:** Global variables are declared outside of any function and have global scope, making them accessible throughout the entire program. They are allocated memory when the program starts and persist until the program terminates. Global variables can be accessed and modified by any function in the program, which can simplify sharing data across different parts of the program. However, excessive use of global variables can make the program harder to understand and maintain.

**Static variables:** Static variables are declared with the static keyword and retain their values across function calls. They have a "static" lifetime, meaning they persist throughout the entire program's execution. Static variables are allocated memory only once, and their values are preserved between function calls. They have either file scope or function scope, depending on where they are declared. When declared within a function, they have local scope, similar to automatic variables, but their values are preserved between function calls. Declaration of static variable as: **static** int gravitationalConstant

**External variables:** External variables are declared with the ‘extern’ keyword and are used to access variables that are defined in other files or modules. They have global scope and can be accessed by multiple files in a program. The extern keyword tells the compiler that the variable is defined elsewhere, and it is used for sharing data between different files. External variables are useful when multiple files need to access and share the same data, providing a way to make the variables accessible and avoid redefinition. Declaration of extern variable as: **extern** int gravitationalConstant (stored in one file and used in multiple)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SET – 2 Answers**

**4 Answer:**

**Array:** an array is a collection of elements of the same type, stored in contiguous memory locations. It provides a way to store multiple values under a single name. Each element in the array is accessed by its index, which represents its position in the array. In C, the array index starts from 0.

* An array is generally declared as follows: datatype arrayName[arraySize];

Here, ‘datatype’ represents the type of elements the array will hold, such as int, float, char, etc. ‘arrayName’ is the name given to the array, and ‘arraySize’ specifies the number of elements the array can hold.

**Declaration and initialization of 1-Dimensional array:**

The general way of declaring a 1-D array is as follows:

* datatype arrayName[arraySize] = {value1, value2, ..., valueN};

the values of the array initialized by grouping the in curly brackets {} separated by commas. **Example:**

#include <stdio.h>

int main() {

int numbers[5] = {10, 20, 30, 40, 50}; /\*Declaration & Initialization \*/

/\* Access and print the array elements \*/

for (int i = 0; i < 5; i++) {

printf(" %d element of the array : %d\n", i, numbers[i]);

}

return 0;

}

In this example, we declare and initialize the numbers array with 5 elements with integer datatype: 10, 20, 30, 40, and 50. Then, we use a for loop to access and print each element of the array. The loop iterates from i = 0 to i < 5, and we use numbers[i] to access the array elements.

**Declaration and initialization of 2-Dimensional array:**

2-D arrays are usually useful to store and manipulate data in a tabular format, such as matrices or tables. The general way of declaring and initializing 2-D array is as follows:

* datatype arrayName[rowSize][columnSize] = {{row1}, {row2}, …, {rowN}};

here, each row itself behaves like 1-D array of size equals to the column size

**Example:** The following example takes 4 subject grades of 3 students and prints them out student-wise.

#include <stdio.h>

int main(){

int grades[3][4]; /\* Declare a 2D array for grades\*/

/\* Input grades for each student and subject\*/

printf("Enter grades for each student and subject:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 4; j++) {

printf("Enter grade for student %d in subject %d: ", i + 1, j + 1);

scanf("%d", &grades[i][j]);

}

}

/\* Display the grades \*/

printf("\nGrades:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 4; j++) {

printf("Grade for student %d in subject %d: %d\n", i + 1, j + 1, grades[i][j]);

}

}

return 0;

}

In this program, we first declare a 2D array grades to store the student grades. We then use nested loops to input the grades for each student and subject. Afterward, we display the entered grades using another set of nested loops. When the program is run, it will prompt to enter the grades for each student and subject. Once the grades are provided, they will be displayed as output on the screen.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5 Answer:**

**5a)** A pointer is a variable that holds the memory address of another variable. It allows us to indirectly access and manipulate the value of the variable it points to. Pointers are powerful tools in C programming that allow us to efficiently manipulate and work with memory addresses. Understanding pointers is essential for tasks like dynamic memory allocation, passing variables by reference, and working with arrays and complex data structures. To declare a pointer variable, we use the asterisk (\*) symbol. For example:

int \*p; /\* declaration of a pointer\*/

This declares a pointer named p that can hold the memory address of an integer variable. The asterisk (\*) is placed before the variable name tells the compiler that it is a pointer.

To initialize a pointer, we assign it the address of another variable. This is done using the address-of operator (&). For example:

int num = 10;

int \*p = &num; /\* initializing a pointer \*/

In the above example, we declare an integer variable num and assign it the value 10. Then, we declare a pointer p and initialize it with the address of the num variable using the & operator. Now, the pointer p points to the memory location of the num variable.

**5b)** An example C program that demonstrates the use of the indirection operator (\*) to access the value pointed to by a pointer. The program calculates the area of a circle from the radius input given by the user, using pointers:

#include <stdio.h>

float calculateArea(float \*radius) {

float pi = 3.14;

return pi \* (\*radius) \* (\*radius);

}

int main() {

float radius;

printf("Enter the radius of the circle: ");

scanf("%f", &radius); /\* taking input from the user \*/

float \*ptr = &radius; /\* ptr declaration and initialization \*/

float area = calculateArea(ptr); /\* calling function with pointer as argument\*/

printf("The area of the circle with radius %f is %f\n", radius, area);

return 0;

}

In the above program, we first declare the calculateArea() function that takes a pointer to a float variable as a parameter. Inside the function, we declare a local variable pi to store the value of pi (approximately 3.14). By using the indirection operator (\*) on the radius pointer, we access the value pointed to by the pointer and use it in the area calculation formula. In the main() function, we declare a float variable radius to store the user input for the radius of the circle. We then declare a pointer ptr and initialize it with the address of the radius variable using the address-of operator (&). This allows us to pass the address of the radius variable to the calculateArea() function. Next, we call the calculateArea() function, passing the ptr pointer as an argument. The function calculates the area of the circle using the value pointed to by the radius pointer and returns the result. Finally, we display the radius and calculated area of the circle using the printf() function.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6 Answer:**

**6a) Structure:** A structure in C is a user-defined data type that allows you to combine different variables of different data types under a single name. It provides a way to group related data together.

**General syntax:**

struct structName {

dataType member1;

dataType member2;

.......

};

explanation : the structName is the name given to the structure, and member1, member2, and so on, represent the variables or members of the structure, each with its own data type.

To create variables of the structure type, we use the struct keyword followed by the structure name and variable names as shown: struct structName variableName;

To access the members of a structure, you use the dot operator (.) between the structure variable name and the member name. General syntax

variableName.memberName = value;

**Example**: we will create a struct called car that has the data of car parameters as members. We will create a car struct for a particular car and manipulate the values.

struct Car {

char brand[50];

int year;

float price;

};

The above is a structure called Car that has three members: brand, year, and price. The brand member is an array of characters (a string) with a maximum length of 50 characters. The year member is an integer, and the price member is a floating-point number.

to create a structure variable named car1 of type Car and access its members:

struct Car car1;

car1.year = 2022;

strcpy(car1.brand, "Toyota");

car1.price = 2500000;

we declare a structure variable car1 of type Car. We then assign values to its members using the dot operator. For example, we assign the year 2022 to car1.year, the brand "Toyota" to car1.brand, and the price 2500000 to car1.price and to access year of car1 we can simply use car1.year.

**6b)** Example of a C program that uses structures to store and print employee details:

#include <stdio.h>

struct Employee {

char name[50];

int age;

float salary;

};

void printEmployeeDetails(struct Employee emp) {

printf("Employee Details:\n");

printf("Name: %s\n", emp.name);

printf("Age: %d\n", emp.age);

printf("Salary: %.2f\n", emp.salary);

}

int main() {

struct Employee emp1;

printf("Enter employee name: ");

scanf("%s", emp1.name);

printf("Enter employee age: ");

scanf("%d", &emp1.age);

printf("Enter employee salary: ");

scanf("%f", &emp1.salary);

printEmployeeDetails(emp1);

return 0;

}

In the above program, we define a structure called Employee that has three members: name, age, and salary. The printEmployeeDetails() function takes an Employee structure as a parameter and prints the employee's name, age, and salary.

In the main() function, we declare an emp1 variable of type Employee to store the details of the employee. We prompt the user to input the employee's name, age, and salary using scanf(), and store the values in the corresponding members of emp1.Finally, we call the printEmployeeDetails() function, passing emp1 as an argument, to print the employee's details on the screen. When the program is run, it will ask to enter the employee's name, age, and salary. After the input is provided, it will display the entered employee details on the screen using the printEmployeeDetails() function**.**